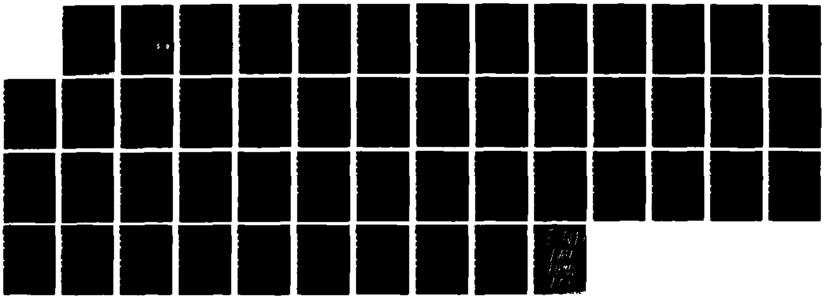
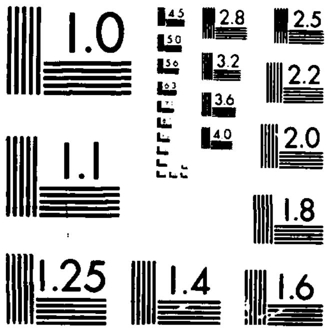
AD-A193 486 ADA (TRADE NAME) COMPILER VALIDATION SUMMARY REPORT 1/1
SYSTEAM KG DR WINTERS (U) INFORMATION SYSTEMS AND
TECHNOLOGY CENTER W-P AFB OH ADA VALI 11 FEB 87
UNCLASSIFIED AVF-VSR-010 F/G 12/5 NN




1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

2

n Data Entered)

AD-A193 486

TION PAGE

	READ INSTRUCTIONS BEFORE COMPLETEING FORM
2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: SYSTEAM KG Dr Winterstein SYSTEAM Germa MoD VAX-VMS Ada Comp, Ver.VI.6 VAX 8500	5. TYPE OF REPORT & PERIOD COVERED 02-11-87 to 02-11-88
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) IABG m.b.H, Dept SZT	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS IABG m.b.H., Dept. SZT Einsteinstrasse 20, D-8012 Ottobrunn, West Germany	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081ASD/SIOL	12. REPORT DATE 02-11-87
	13. NUMBER OF PAGES 47
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* IABG m.b.H.	15. SECURITY CLASS *(of this report)* UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT *(of this Report)* Approved for public release; distribution unlimited.	
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20. If different from Report)* UNCLASSIFIED	
18. SUPPLEMENTARY NOTES	

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See Attached.

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the SYSTEAM German MoD VAX-VMS Ada Compiler, Version V1.6, using Version 1.8 of the Ada Compiler Validation Capability (ACVC). The SYSTEAM German MoD VAX-VMS Ada Compiler is hosted on a VAX 8500 operating under VMS, Version 4.5. Programs processed by this compiler may be executed on a VAX 8500 operating under VMS Version 4.5.

On-site testing was performed 87-02-09 through 87-02-11 at SYSTEAM KG, Karlsruhe, under the direction of the IABG m.b.H., Dept SZT (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2138 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 242 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2138 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 33 of the processed tests determined to be inapplicable. The remaining 2105 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTA
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	94	222	298	244	161	97	137	261	124	32	218	217	210
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	22	103	122	3	0	0	2	1	6	0	0	16	27
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	1
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	239

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

* Ada is a registered trademark of the United States Government (Ada Joint Program Office).

AVF Control Number: AVF-VSR-010

Ada[*] COMPILER VALIDATION SUMMARY REPORT: SYSTEAM KG Dr. Winterstein SYSTEAM German MoD VAX-VMS Ada Compiler, Version V1.6 VAX 8500

Completion of On-Site Testing:
87-02-11

Prepared By:
IABG m.b.H., Dept SZT
Einsteinstrasse 20
D-8012 Ottobrunn
West Germany

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

* Ada is a registered trademark of the United States Government (Ada Joint Program Office).

Ada* Compiler Validation Summary Report:

Compiler Name: SYSTEAM German MoD VAX-VMS Ada Compiler, Version V1

Host:	Target:
VAX 8500 under VMS, Version 4.5	VAX 8500 under VMS, Version 4.5

Testing Completed 87-02-11 Using ACVC 1.8

This report has been reviewed and is approved.

[illegible]

IABG m.b.H., Dept SZT
Dr. H. Hummel
Einsteinstrasse 20
D-8012 Ottobrunn
West Germany

[illegible]

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

* Ada is a registered trademark of the United States Government (Ada Joint Program Office).

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the SYSTEAM German MoD VAX-VMS Ada Compiler, Version V1.6, using Version 1.8 of the Ada Compiler Validation Capability (ACVC). The SYSTEAM German MoD VAX-VMS Ada Compiler is hosted on a VAX 8500 operating under VMS, Version 4.5. Programs processed by this compiler may be executed on a VAX 8500 operating under VMS Version 4.5.

On-site testing was performed 87-02-09 through 87-02-11 at SYSTEAM KG, Karlsruhe, under the direction of the IABG m.b.H., Dept SZT (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2138 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 242 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2138 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 33 of the processed tests determined to be inapplicable. The remaining 2105 tests were passed.

The results of validation are summarized in the following table:

RESULT						CHAPTER							TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	94	222	298	244	161	97	137	261	124	32	218	217	2105
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	22	103	122	3	0	0	2	1	6	0	0	16	275
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

---

* Ada is a registered trademark of the United States Government (Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies -- for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from characteristics of particular operating systems, hardware, or implementation strategies. All of the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any unsupported language constructs required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted, under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 87-02-09 through 87-02-11 at SYSTEAM KG, Karlsruhe.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. /552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

IABG m.b.H., Dept SZT
Einsteinstrasse 20
D-8012 Ottobrunn
West Germany

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Validation Procedures and Guidelines, Ada Joint Program Office, Jan 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host	The computer on which the compiler resides.
Inapplicable test	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	A test found to be incorrect and not used to check conformity to the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters -- for example, the number of identifiers permitted in a compilation or the number of units in a library -- a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time -- that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values -- for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: SYSTEAM German MoD VAX-VMS Ada Compiler, Version V1.

ACVC Version: 1.8

Certificate Expiration Date: 88-05-13

" #: 870211I1.08036

Host Computer:

Machine:	VAX 8500
Operating System:	VMS Version 4.5
Memory Size:	20 MB

Target Computer:

Machine:	VAX 8500
Operating System:	VMS Version 4.5
Memory Size:	20 MB

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

. Capacities.

The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

. Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

. Predefined types.

This implementation supports the additional predefined types SHORT_INTEGER and SHORT_FLOAT in the package STANDARD. (See tests B86001C and B86001D.)

. Based literals.

An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array objects are declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.) All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Functions.

An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in the same immediate scope, or it may reject the function declaration. If it accepts the function declarations, the use of the enumeration literal's identifier denotes the function. This implementation rejects the declarations. (See test E66001D.)

- Representation clauses.

  The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation accepts 'SIZE and 'STORAGE_SIZE for tasks, 'STORAGE_SIZE for collections, and 'SMALL clauses. Enumeration representation clauses, including those that specify non-contiguous values, appear to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

- Pragmas.

  The pragma INLINE is not supported for procedures. The pragma INLINE is not supported for functions. (See tests CA3004E and CA3004F.)

- Input/output.

  The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO can not be instantiated with an unconstrained array type. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

  An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and can be created in IN_FILE mode. (See test EE3102C.)

  More than one internal file can be associated with each external file for text I/O for reading only. (See tests CE3111A..E (5 tests).)

  More than one internal file can be associated with each external file for sequential I/O for reading only. (See tests CE2107A..F (6 tests).)

  More than one internal file can be associated with each external file for direct I/O for reading only. (See tests CE2107A..F (6 tests).)

  Temporary sequential files are not given a name. Temporary direct files are not given a name. (See tests CE2108A and CE2108C.)

- Generics.

  Generic subprogram declarations and bodies can be compiled in separate compilations. (See test CA2009F.) Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C and BC3205D.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of SYSTEAM German MoD VAX-VMS Ada Compiler was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 275 tests were inapplicable to this implementation, and that the 2105 applicable tests were passed by the implementation. There were no failed tests.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	69	864	1100	17	11	44	2105
Failed	0	0	0	0	0	0	0
Inapplicable	0	3	268	0	2	2	275
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	94	222	298	244	161	97	137	261	124	32	218	217	2105
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	22	103	122	3	0	0	2	1	6	0	0	16	275
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B33203C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B37401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 275 tests were inapplicable for the reasons indicated:

. C24113D..E (2 tests) are inapplicable because they contain lines which are longer than MAX_IN_LEN.

. C34001E, B52004D, B55B09C, and C55B07A use LONG_INTEGER which is not supported by this compiler.

. C34001G and C35702B use LONG_FLOAT which is not supported by this compiler.

. B86001D is inapplicable because there is no valid substitution for the macro $NAME.

. C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

. C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATION's base type. This is not the case for this implementation.

. CA3004E, EA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

- CA3004F, EA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

- CE2108A, CE2108C and CE3112A are inapplicable because temporary files have no name.

- CE2107B..E (4 tests), CE2110B, CE2111D, CE2111H, CE3111B..E (4 tests) and CE3114B are inapplicable because multiple internal files can only be associated with the same external file for reading. The proper exception is raised when multiple access is attempted.

- CE2401D is inapplicable because a use_error is raised for create.

- The following 242 tests require a floating-point accuracy that exceeds the maximum of 9 supported by the implementation:

  C24113F..Y (20 tests)
  C35705F..Y (20 tests)
  C35706F..Y (20 tests)
  C35707F..Y (20 tests)
  C35708F..Y (20 tests)
  C35802F..Y (20 tests)
  C45241F..Y (20 tests)
  C45321F..Y (20 tests)
  C45421F..Y (20 tests)
  C45424F..Y (20 tests)
  C45521F..Z (21 tests)
  C45621F..Z (21 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for 6 Class B tests.

B97101E	BC10AEB	BC3204B
BC3204D	BC3205B	BC3205C
BC3205D		

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by the SYSTEAM German MoD VAX-VMS Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the SYSTEAM German MoD VAX-VMS Ada Compiler using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX 8500 host/target operating under VMS, Version 4.5.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring splits during the prevalidation testing were included in their split form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled on the VAX 8500, and all executable tests were linked and executed on the target. Results were printed from the target computer.

The compiler was tested using command scripts provided by SYSTEAM KG Dr. Winterstein and reviewed by the validation team. The following options were in effect for testing:

```
for the B-tests
    ADAV16: compiler test-name options=list=>on
for all others
    ADAV16: compiler test-name
```

Tests were compiled, linked, and executed (as appropriate) using a single host computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

This is a short example of the command script used to run the tests.

```
_DUAO:[USER.ACVC.ACVC15PRE]RUN2A.COM;1

$! p1 : compiler version
$! p2 : additional parameter for compiler
$ version = p1
$ env = f$environment("PROCEDURE")
$ acvcversion = f$parse(env,,,"DEVICE")  + f$parse(env,,,"DIRECTORY")
$ acvc  = acvcversion - "]" + ".-]"
$ Achap = acvcversion - "]" + ".atests.a2]"
$ Bchap = acvcversion - "]" + ".btests.b2]"
$ Cchap = acvcversion - "]" + ".ctests.c2]"
$ Dchap = acvcversion - "]" + ".dtests.d2]"
$ Echap = acvcversion - "]" + ".etests.e2]"
$ Lchap = acvcversion - "]" + ".ltests.l2]"
$ acvcsupport = acvcversion - "]" + ".support]"
$!
$ @'version'createlib
$ @'version'compile 'acvcsupport'repspec
$ @'version'compile 'acvcsupport'repbody
$!
$ @'version'compile 'Achap'A21001A.ADA  'p2'
$ @'acvc'linkandgo A21001A 'version'
$ @'version'compile 'Achap'A22002A.ADA  'p2'
$ @'acvc'linkandgo A22002A 'version'
$ veri = f$verify(0)
$!**************************************************************************
$! Command procedure to link and execute one module of
$! the acvc's.
$! p1 = name of test
$! p2 = compiler version
$! p3 = additional parameters for 'version'link
$!**************************************************************************
$!
$ testname    = p1
$ version     = p2
$!
$ on error then goto error_exit
$ @'version'link 'testname' 'testname' 'p3' debug=off
$ create 'testname'.res
$ define/user sys$output 'testname'.res
$ run 'testname'
$ delete 'testname'.exe;
$ veri = f$verify(veri)
$ exit
$ error_exit :
$ write sys$output ">>> 'linkandgo' not terminated normally "
$ if f$search ("[...lib.obj]*.*") .nes. "" then delete [.adalib.obj]*.*
$ veri = f$verify(veri)
$ exit
```

APPENDIX A

DECLARATION OF CONFORMANCE

Compiler Implementer:	SYSTEAM KG Dr. Winterstein
Ada Validation Facility:	IABG
Ada Compiler Validation Capability (ACVC)	Version: 1.8

Base Configuration

Base Compiler Name:	SYSTEAM/German MoD VAX/VMS Ada Compiler Version 1.6	
Host Architecture	ISA: VAX 8500	OS&VER No: VMS 4.5
Target Architecture	ISA: VAX 8500	OS&VER No: VMS 4.5

Implementer's Declaration

I, the undersigned, representing SYSTEAM KG have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that SYSTEAM KG is the owner of record of the Ada language compiler listed above and, as such,is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

[illegible]

Dr. Winterstein Karlsruhe, den 06.02.87

Owner's Declaration

I, the undersigned, representing SYSTEAM KG take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compiler listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compiler and concur with the contents.

[illegible]

Dr. Winterstein Karlsruhe, den 06.02.87

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation- dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the SYSTEAM German MoD VAX-VMS Ada Compiler, Version V1.6, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). The specification of the package STANDARD is also included in this appendix.

```
PACKAGE standard IS

   TYPE boolean IS (false, true);

   TYPE short_integer IS RANGE - 32_768 .. 32_767;

   TYPE integer IS RANGE - 2_147_483_648 .. 2_147_483_647;

   TYPE short_float IS DIGITS 6 RANGE
      - 16#0.7FFF_FF8#E+32 ..
        16#0.7FFF_FF8#E+32;

   TYPE float IS DIGITS 9 RANGE
      - 16#0.7FFF_FFFF_FFFF_FF8#E+32 ..
        16#0.7FFF_FFFF_FFFF_FF8#E+32;

   TYPE character IS

      (nul, soh, stx, etx,       eot, enq, ack, bel,
       bs , ht , lf , vt ,       ff , cr , so , si ,
       dle, dc1, dc2, dc3,       dc4, nak, syn, etb,
       can, em , sub, esc,       fs , gs , rs , us ,

       ' ', '!', '"', '#',       '$', '%', '&', ''',
       '(', ')', '*', '+',       ',', '-', '.', '/',
       '0', '1', '2', '3',       '4', '5', '6', '7',
       '8', '9', ':', ';',       '<', '=', '>', '?',

       '@', 'A', 'B', 'C',       'D', 'E', 'F', 'G',
       'H', 'I', 'J', 'K',       'L', 'M', 'N', 'O',
       'P', 'Q', 'R', 'S',       'T', 'U', 'V', 'W',
       'X', 'Y', 'Z', '[',       '\', ']', '^', '_',

       '`', 'a', 'b', 'c',       'd', 'e', 'f', 'g',
       'h', 'i', 'j', 'k',       'l', 'm', 'n', 'o',
       'p', 'q', 'r', 's',       't', 'u', 'v', 'w',
       'x', 'y', 'z', '{',       '|', '}', '~', del);

   FOR character USE -- ASCII characters without holes
      (0  , 1  , 2  , 3  , 4  , 5  , 6  , 7  ,
       8  , 9  , 10 , 11 , 12 , 13 , 14 , 15 ,
       16 , 17 , 18 , 19 , 20 , 21 , 22 , 23 ,
       24 , 25 , 26 , 27 , 28 , 29 , 30 , 31 ,
       32 , 33 , 34 , 35 , 36 , 37 , 38 , 39 ,
       40 , 41 , 42 , 43 , 44 , 45 , 46 , 47 ,
       48 , 49 , 50 , 51 , 52 , 53 , 54 , 55 ,
       56 , 57 , 58 , 59 , 60 , 61 , 62 , 63 ,
       64 , 65 , 66 , 67 , 68 , 69 , 70 , 71 ,
       72 , 73 , 74 , 75 , 76 , 77 , 78 , 79 ,
       80 , 81 , 82 , 83 , 84 , 85 , 86 , 87 ,
        88 , 89 , 90 , 91 , 92 , 93 , 94 , 95 ,
        96 , 97 , 98 , 99 , 100, 101, 102, 103,
        104, 105, 106, 107, 108, 109, 110, 111,
        112, 113, 114, 115, 116, 117, 118, 119,
        120, 121, 122, 123, 124, 125, 126, 127);

   PACKAGE asoii IS

      -- Control oharaoters:

      nul        : CONSTANT oharaoter := nul;
      soh        : CONSTANT oharaoter := soh;
      stx        : CONSTANT oharaoter := stx;
      etx        : CONSTANT oharaoter := ext;
      eot        : CONSTANT oharaoter := eot;
      enq        : CONSTANT oharaoter := enq;
      aok        : CONSTANT oharaoter := aok;
      bel        : CONSTANT oharaoter := bel;
      bs         : CONSTANT oharaoter := bs;
      ht         : CONSTANT oharaoter := ht;
      lf         : CONSTANT oharaoter := lf;
      vt         : CONSTANT oharaoter := vt;
      ff         : CONSTANT oharaoter := ff;
      or         : CONSTANT oharaoter := or;
      so         : CONSTANT oharaoter := so;
      si         : CONSTANT oharaoter := si;
      dle        : CONSTANT oharaoter := dle;
      dol        : CONSTANT oharaoter := dol;
      do2        : CONSTANT oharaoter := do2;
      do3        : CONSTANT oharaoter := do3;
      do4        : CONSTANT oharaoter := do4;
      nak        : CONSTANT oharaoter := nak;
      syn        : CONSTANT oharaoter := syn;
      etb        : CONSTANT oharaoter := etb;
      oan        : CONSTANT oharaoter := oan;
      em         : CONSTANT oharaoter := em;
      sub        : CONSTANT oharaoter := sub;
      eso        : CONSTANT oharaoter := eso;
      fs         : CONSTANT oharaoter := fs;
      gs         : CONSTANT oharaoter := gs;
      rs         : CONSTANT oharaoter := rs;
      us         : CONSTANT oharaoter := us;
      del        : CONSTANT oharaoter := del;

      -- Other oharaoters:

      exolam     : CONSTANT oharaoter := '!';
      quotation  : CONSTANT oharaoter := '"';
```

```
    sharp       : CONSTANT character := '#';
    dollar      : CONSTANT character := '$';
    percent     : CONSTANT character := '%';
    ampersand   : CONSTANT character := '&';
    colon       : CONSTANT character := ':';
    semicolon   : CONSTANT character := ';';
    query       : CONSTANT character := '?';
    at_sign     : CONSTANT character := '@';
    l_bracket   : CONSTANT character := '[';
    back_slash  : CONSTANT character := '\';
    r_bracket   : CONSTANT character := ']';
    circumflex  : CONSTANT character := '^';
    underline   : CONSTANT character := '_';
    grave       : CONSTANT character := '`';
    l_brace     : CONSTANT character := '{';
    bar         : CONSTANT character := '|';
    r_brace     : CONSTANT character := '}';
    tilde       : CONSTANT character := '~';

    -- Lower case letters:

    lc_a        : CONSTANT character := 'a';
    lc_b        : CONSTANT character := 'b';
    lc_c        : CONSTANT character := 'c';
    lc_d        : CONSTANT character := 'd';
    lc_e        : CONSTANT character := 'e';
    lc_f        : CONSTANT character := 'f';
    lc_g        : CONSTANT character := 'g';
    lc_h        : CONSTANT character := 'h';
    lc_i        : CONSTANT character := 'i';
    lc_j        : CONSTANT character := 'j';
    lc_k        : CONSTANT character := 'k';
    lc_l        : CONSTANT character := 'l';
    lc_m        : CONSTANT character := 'm';
    lc_n        : CONSTANT character := 'n';
    lc_o        : CONSTANT character := 'o';
    lc_p        : CONSTANT character := 'p';
    lc_q        : CONSTANT character := 'q';
    lc_r        : CONSTANT character := 'r';
    lc_s        : CONSTANT character := 's';
    lc_t        : CONSTANT character := 't';
    lc_u        : CONSTANT character := 'u';
    lc_v        : CONSTANT character := 'v';
    lc_w        : CONSTANT character := 'w';
    lc_x        : CONSTANT character := 'x';
    lc_y        : CONSTANT character := 'y';
    lc_z        : CONSTANT character := 'z';

END ascii;
```

```
   -- Predefined subtypes:

   SUBTYPE natural  IS integer RANGE 0 .. integer'last;
   SUBTYPE positive IS integer RANGE 1 .. integer'last;

   -- Predefined string type:

   TYPE string IS ARRAY (positive RANGE <>) OF oharaoter;

   PRAGMA paok (string);

   TYPE duration IS DELTA 2#1.0#E-14 RANGE
      - 131_072.0 .. 131_071.999_938_964_843_75;

   -- The predefined exoeptions:

   oonstraint_error : EXCEPTION;
   numerio_error    : EXCEPTION;
   program_error    : EXCEPTION;
   storage_error    : EXCEPTION;
   tasking_error    : EXCEPTION;

END standard;
```

The following predefined library units are inoluded in eaoh newly oreated program library:

```
   The paokage SYSTEM
   The paokage CALENDAR
   The generio prooedure UNCHECKED_DEALLOCATION
   The generio funotion UNCHECKED_CONVERSION
   The paokage IO_EXCEPTIONS
   The generio paokage SEQUENTIAL_IO
   The generio paokage DIRECT_IO
   The paokage TEXT_IO
   The paokage LOW_LEVEL_IO
```

9 IMPLEMENTATION-DEPENDENT CHARACTERISTICS

This chapter corresponds to Appendix F of the Ada Language Reference Manual, which describes all implementation-dependent characteristics.

9.1 Implementation-Dependent Pragmas

INLINE -
inline inclusion is never done

INTERFACE -
is implemented for ASSEMBLER

SQUEEZE -
takes the same argument as the predefined language pragma PACK and is allowed at the same positions. It causes the compiler to select a representation for the argument-type that needs minimal storage space. By contrast, the pragma PACK only leads to representations which cause components of objects of its argument-types to start on storage-unit-bounds.

SUPPRESS_ALL -
causes that all checks that may raise CONSTRAINT_ERROR at run-time are suppressed; this pragma is only allowed at the start of a compilation before the first compilation unit; it applies to the whole compilation

9.2 Implementation-Dependent Attributes

HEAP_ADDRESS -
applied to an access type yields a value of type ADDRESS (from package SYSTEM). This attribute is only for internal use within the package COLLECTION_MANAGER.

9.3 Specification of the Package SYSTEM

```
PACKAGE system IS

   TYPE address IS PRIVATE;

   TYPE name IS (vax_730, vax_750, vax_780, vax_782);

   system_name   : CONSTANT name := vax_750;
   storage_unit  : CONSTANT := 8;
   memory_size   : CONSTANT := 2 ** 31;
   min_int       : CONSTANT := - 2_147_483_648;
   max_int       : CONSTANT := 2_147_483_647;
   max_digits    : CONSTANT := 9;
   max_mantissa  : CONSTANT := 31;
   fine_delta    : CONSTANT := 2#1.0#E-30;
   tick          : CONSTANT := 0.2E-6;

   SUBTYPE priority IS integer RANGE 0 .. 255;

   SUBTYPE external_address IS string;

   SUBTYPE byte IS integer RANGE 0..255;

   TYPE    long_word IS ARRAY (0..3) OF byte;
   PRAGMA  PACK (long_word);

   FUNCTION convert_address (addr   : external_address)
                             RETURN address;

   FUNCTION convert_address (addr   : address)
                             RETURN external_address;

   FUNCTION convert_address (addr   : long_word)
                             RETURN address;

   FUNCTION convert_address (addr   : address)
                             RETURN long_word;

   FUNCTION "+"             (addr   : address;
                             offset : integer)
                             RETURN address;

PRIVATE

   -- private declarations

END system;
```

External addresses are represented as strings consisting of hexadecimal digits.

Since the type ADDRESS is private, no representation specifications for objects of this type can be given. If representation specifications for addresses are required, objects of type LONG_WORD can be used to hold address values.

Overloaded functions CONVERT_ADDRESS are defined to allow conversion between the different represenations of addresses.

9.4 Restrictions on Representation Clauses

Address clauses are only implemented for objects.

The value given in a specification of small for a fixed point type must be a power of two.

9.5 Conventions for Implementation-Generated Names

There are no implementation-generated names denoting implementation-dependent components.

9.6 Interpretation of Address Clauses

An object for which an address specification is given must not require an initialization (neither explicit nor implicit). Otherwise the program is erroneous.

The object starts at the given address. For objects accessed by a descriptor, the descriptor starts at the given address.

9.7 Restrictions on Unchecked Conversions

If

```
   TARGET'SIZE > SOURCE'SIZE
```

the result value of the unchecked conversion is unpredictable.

9.8 Characteristics of the Input-Output Packages

9.8.1 The NAME Parameter

The string must be a VMS file specification string. The function NAME will return a file specification string (including version number) which is the resultant filename of the file opened or created.

The exception NAME_ERROR is raised if the name parameter is not a legal VMS file specification string; for example, if it contains illegal characters, is too long or is syntactically incorrect. The file specification string must not contain wild cards even if an unique file is specified; otherwise the exception NAME_ERROR is raised.

In an OPEN operation the exception NAME_ERROR is also raised if the specified file does not exist; in a CREATE operation this exception is raised if the NAME string contains an explicit version number and the specified file already exists.

9.8.2 The FORM Parameter

9.8.2.1 The Syntax of the FORM string

```
form_parameter ::=
    [ form_specification { , form_specification } ]

form_specification ::=  keyword => value

keyword ::=  identifier

value   ::=  identifier | string_literal | numeric_literal
```

For identifier, numeric_literal, string_literal see LRM Appendix E. Only an integer literal is allowed as numeric_literal (see LRM 2.4).

The exception USE_ERROR is raised if a given FORM paramter string does not have the correct syntax or if a condition on a single form specification described in the following sections is not fulfilled.

9.8.2.2 General Form Specifications

In the following, the form specifications which are allowed for all files are described.

- ALLOCATION => numeric_literal

This value specifies the number of blocks which are allocated initially; it is only used in a create operation and ignored in an open operation. The value of allocation in the form string returned by the function form specifies the initial allocation size for existing files too.

- EXTENSION => numeric_literal

This value specifies the number of blocks by which a file is extended if necessary; the value 0 means that the RMS default value is taken. For existing files this value is only used for processing between an open and a close operation.

For details see the VAX-11 / RMS Reference Manual.

9.8.3 Text I/O

9.8.3.1 Implementation Dependent Types

The implementation dependent types COUNT and FIELD defined in the package specification of TEXT_IO have the following upper bounds :

COUNT'LAST = 2_147_483_647 (= INTEGER'LAST)

FIELD'LAST = 255

9.8.3.2 Text Files

Text files are represented as sequential files with variable record format. One line is represented as a sequence of one or more records; all records except from the last one have a length of exactly MAX_RECORD_SIZE.

A line terminator is not represented explicitly in the external file; the end of a record which is shorter than MAX_RECORD_SIZE is taken as a line terminator. The value MAX_RECORD_SIZE may be specified by the form string for an output file and it is taken from the external file for an input file; for all files the value 0 stands for the default of 255.

A page terminator is represented as record consisiting of a single ASCII.FF. A record of length zero is assumed to precede a page terminator if the record before the page terminator is another page terminator or a record of length MAX_RECORD_SIZE; this implies that a page terminator is preceded by a line terminator in all cases.

A file terminator is not represented explicitly in the external file; the end of the file is taken as a file terminator. A page terminator is assumed to precede the end of the file if there is not explicitly one as the last record of the file. For input from a terminal, a file terminator is represented as ASCII.SUB (= CTRL Z).

In the following, the form specifications which are only allowed for text files are described.

Only for output files :

- MAX_RECORD_SIZE => numeric_literal

This value specifies the maximum length of a record in the external file. Each record which is not the last record of a line has exactly this maximum record size. The value must be in the range from 1 up to 255. If a file is created with a maximum record size different from the default of 255, the external file gets the specified value as maximum record size; otherwise, the external file gets the value 0 as maximum record size (no explicit maximum record size). If the value is specified for an existing file it must be conform with the value of the external file.

- END_OF_FILE

If the keyword END_OF_FILE is specified for an existing file in an open for an output file then the file is opened at the end of the file; i.e. the existing file is extended and not rewritten. This keyword is only allowed for an output file; it only has an effect in an open operation and is ignored in a create.

Only for input files :

- PROMPTING => string_literal

This string is output on the terminal before an input record is read if the input file is associated with a terminal; otherwise this form specification is ignored.

The default form string for an input text file is :

```
   "ALLOCATION => 3, EXTENSION => 0, PROMPTING => """"      "
```

The default form string for an output text file is :

```
    "ALLOCATION => 3, EXTENSION => 0, MAX_RECORD_SIZE => 255"
```

- CHARACTER_IO

In addition to the input/output facilities with record structured external files, another form of input/output is provided for text files: It is possible to transfer single characters from/to a terminal device. This form of input/output is specified by the keyword CHARACTER_IO in the form string. If character i/o is specified, no other form specification is allowed and the file name must denote a terminal device.

For an infile the external file (associated with a terminal) is considered to contain a single line. An ASCII.SUB (= CTRL Z) character represents an line terminator followed by a page terminator followed by a file terminator. Arbitrary characters (including all control characters except from ASCII.SUB) may be read; a character read is not echoed to the terminal.

For an outfile, arbitrary characters (including all control characters and escape sequences) may be written on the external file (terminal). A line terminator is represented as ASCII.CR followed by ASCII.LF, a page terminator is represented as ASCII.FF and a file terminator is not represented on the external file.

9.8.3.3 Standard Files

The standard input (resp. output) file is associated with SYS$INPUT (resp. SYS$OUTPUT). If a program reads from the standard input file, the logical name SYS$INPUT must denote an existing file. If a program writes to the standard output file, a file with the logical name SYS$OUTPUT is created if no such file exists; otherwise the existing file is extended.

The qualifiers /INPUT and /OUTPUT may be used for the VMS RUN command to associate VMS files with the standard files of TEXT_IO.

The name and form strings for the standard files are :

```
standard_input  :  NAME => "SYS$INPUT:"
                   FORM => "PROMPTING => ""*""     "

standard_output :  NAME => "SYS$OUTPUT:"
                   FORM => "MAX_RECORD_SIZE => 255"
```

9.8.4 Sequential and Direct Files

Sequential and direct files are represented by RMS sequential, relative or indexed files with fixed-length or variable-length records. Each element of the file is stored in one record.

9.8.4.1 Restrictions Concerning the ELEMENT TYPE

input/output of access types is not defined.

- the attribute ADDRESS applied to an object of the element type must specify the start address of the value of the object (not the address of a descriptor).

- input/output is not possible for an object whose start address is not byte aligned (may only occur if a representation specification is given).

- the attribute SIZE applied to an object of the element type must deliver the number of bits allocated contiguously in the memory for the object; this value must be a multiple of SYSTEM.STORAGE_UNIT. For example, objects of record types with dynamic array components are not stored contiguously.

- if a fixed record format is used, all objects to be input or output must have the same size (ELEMENT_TYPE'SIZE).

- input/output of elements of an unconstrained array type is only possible for files with variable-length records.

- for RMS sequential [relative] files the size of an object to be input or output must not be greater than 32767 [16383].

9.8.4.2 Sequential Files

A sequential file is represented by a RMS sequential file with either fixed-length or variable-length records which may be specified by the form parameter.

- MAX_RECORD_SIZE => numeric_literal

This value specifies the maximum record size in bytes; the value 0 indicates that there is no limit. This form specification is only allowed for files with variable record format. If the value is specified for an existing file, it must agree with the value of the external file. For files with fixed-length records, the maximum record size equals ELEMENT_TYPE'SIZE / SYSTEM.STORAGE_UNIT.

- RECORD_FORMAT => VARIABLE | FIXED

This form specification is used to specify the record format. If the format is specified for an existing file, it must equal the format of the external file.

- END_OF_FILE

If the keyword END_OF_FILE is specified for an existing file in an open for an output file, then the file is opened at the end of the file; i.e. the existing file is extended and not rewritten. This keyword is only allowed for an output file; it only has an effect in an open operation and is ignored in a create.

The default form string for a sequential file is :

```
"ALLOCATION    => 3,        EXTENSION       => 0, " &
"RECORD_FORMAT => VARIABLE, MAX_RECORD_SIZE => 0  "
```

9.8.4.3 Direct Files

The implementation dependent type COUNT defined in the package specification of DIRECT_IO has an upper bound of :

COUNT'LAST = 2_147_483_647 (= INTEGER'LAST)

Direct files are represented by RMS sequential files with fixed-length records or by relative or indexed files with either fixed-length or variable-length records. For indexed files, the record index is stored as unsigned four bytes binary value in the first four bytes of each record. If not explicitly specified, the maximum record size equals ELEMENT_TYPE'SIZE / SYSTEM.STORAGE_UNIT.

- BUCKET_SIZE => numeric_literal

This value specifies the number of blocks (one block is 512 bytes) for one bucket; the value 0 means that the value is evaluated by RMS to the minimal number of blocks which is necessary to contain one record. The value must be in the range from 0 up to 32. This form specification is only allowed for relative or indexed files. If the value is specified for an existing file it must agree with the value of the external file.

- MAX_RECORD_SIZE => numeric_literal

This value specifies the maximum record size in bytes. The value 0 which indicates that there is no limit is only allowed for indexed files. A positive value must be greater or equal to ELEMENT_TYPE'SIZE / SYSTEM.STORAGE_UNIT. This form specification is only allowed for files with variable record format. If the value is specified for an existing file it must agree with the value of the external file.

- RECORD_FORMAT => VARIABLE | FIXED

This form specification is used to specify the record format. If the format is specified for an existing file it must equal the format of the external file.

- ORGANIZATION => INDEXED | RELATIVE | SEQUENTIAL

This form specification is used to specify the file organization. If the organization is specified for an existing file it must equal the organization of the external file.

The default form string for a direct file is :

```
   "ALLOCATION   => 3,           EXTENSION     => 0,   " &
   "ORGANIZATION => SEQUENTIAL,  RECORD_FORMAT => FIXED"
```

9.8.5 General Limitations

The total number of open files (including the two standard files) must not be greater than 18. An attempt to exceed this limit raises the exception USE_ERROR.

9.8.6 File Sharing

The only form of file sharing which is allowed is shared reading. If two or more files are associated with the same external file at one time (regardless of whether these files are declared in the same program or task), all of these (internal) files must be opened with the mode IN_FILE. An attempt to open one of these files with another mode than IN_FILE will raise the exception USE_ERROR.

Files assooiated with terminal devioes (whioh is only legal for text files) are exoepted from this restriotion. Suoh files may be opened with an arbitrary mode at the same time and assooiated with the same terminal devioe.

9.8.7 Exoeptions in Input-Output

Besides the situations desoribed in 9.8.1 and 9.8.2 under whioh NAME_ERROR and USE_ERROR may be raised, in the following additional oonditions are listed under whioh one of the exoeptions NAME_ERROR, USE_ERROR, DEVICE_ERROR or DATA_ERROR is raised.

The exoeption USE_ERROR is raised if the oharaoteristios of the external file are not appropriate for the file type; for example, if the reoord size of a file with fixed-length reoords does not oorrespond to the size of the element type of a direot_io or sequential_io file. USE_ERROR is also raised if the funotion NAME is applied to a temporary file.

In general it is only guaranteed that a file whioh is oreated by an Ada program may be reopened by another program if the file types and the form strings are the same.

The exoeption DEVICE_ERROR is never raised. Instead of this exoeption the exoeption USE_ERROR is raised whenever an error oocurred during an operation of the underlying RMS system. This may happen if an internal error was deteoted, an operation is not possible for reasons depending on the file or devioe oharaoteristios, a size restriotion is violated, a oapaoity limit is exoeeded or for similar reasons.

The exoeption DATA_ERROR is raised by the prooedure READ if the size of the element in the external file to be read differs from the storage size of the given variable; this may only happen if a variable reoord size is used. This exoeption is raised too if an element with the speoified position in a direot file does not exist; this is only possible if the file is assooiated with a relative or an indexed file.

In general, the exoeption DATA_ERROR is not raised by the prooedure READ if the element read is not a legal value of the element type.

9.8.8 Specification of the Package LOW LEVEL IO

```
PACKAGE low_level_io IS

   TYPE device_type IS (null_device);

   TYPE data_type IS
      RECORD
         NULL;
      END RECORD;

   PROCEDURE send_control    (device : device_type;
                              data   : IN OUT data_type);

   PROCEDURE receive_control (device : device_type;
                              data   : IN OUT data_type);

END low_level_io;
```

9.9 Requirements for Main Programs

The main program must be a parameterless library procedure.

9.10 Specification of the Package COLLECTION MANAGER

```
GENERIC
   TYPE elem IS PRIVATE;
   TYPE acc  IS ACCESS elem;
   size : integer := 100;
PACKAGE collection_manager IS

   PROCEDURE mark;

      -- Mark the heap of type ACC

   PROCEDURE release;

      -- Deallocate all objects on the heap of ACC which were
      -- allocated after the last MARK operation for that heap.
      -- RELEASE without previous MARK raises CONSTRAINT_ERROR

   PROCEDURE reset;

      -- Deallocate all objects on the heap of ACC

END collection_manager;
```

The difference between the number of calls of the procedures MARK and RELEASE must be in the range 0 .. SIZE. After a call of RESET the effect of all previous calls of MARK and RELEASE is cancelled. The counting of the difference mentioned above starts from 0.

The value delivered by the attribute STORAGE_SIZE applied to the actual type for ACC is meaningless if the Collection Manager is used.

9.11 Other Characteristics

9.11.1 Source Programs

The maximum line length is 80. Longer lines are cut and an error is reported.

9.11.2 Program Library

The maximum number of units contained in a program library is 2_000. The maximum number of imported units for one compilation unit is 63.

9.11.3 ADDRESS and PRIORITY

The package SYSTEM must be named by a with clause of a compilation unit if the predefined attribute ADDRESS or the predefined pragma PRIORITY is used within that unit.

9.11.4 Storage for Tasks

The memory space reserved for a task is 4K byte. If more space is needed by the task, a length clause must be given. The activation of a small task requires about 1.1K byte.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name_and_Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..79=>'A',80=>'1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..79=>'A',80=>'2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..40=>'A',41=>'3',42..80=>
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..40=>'A',41=>'3',42..80=>
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..77=>'0')& "298"
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be the size of the maximum line length.	(1..74=>'0')& "69.0E1"
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	(1..60=>' ')

$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	'"'&lowercase&special_chars&
$FIELD_LAST A universal integer literal whose value is TEXT_IO. FIELD'LAST.	255
$FILE_NAME_WITH_BAD_CHARS An illegal external file namea that either contains invalid characters, or is too long if no invalid characters exist.	abc!odef.dat
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character, or is too long if no wild card character exists.	ABC*DEF.DAT
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in the range of DURATION.	0.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	200_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	x$!yz.dat
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	(1..60=>'A')
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST The universal integer literal	2147483647

	expression whose value is INTEGER'LAST.	
$LESS_THAN_DURATION	A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of	-0.0
$LESS_THAN_DURATION_BASE_FIRST	The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-200_000.0
$MAX_DIGITS	The universal integer literal whose value is the maximum digits supported for floating-point types.	9
$MAX_IN_LEN	The universal integer literal whose value is the maximum input line length permitted by the implementation.	80
$MAX_INT	The universal integer literal whose value is SYSTEM.MAX_INT.	21477483647
$NAME	A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	$NAME
$NEG_BASED_INT	A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16/ FFFFFFFE/
$NON_ASCII_CHAR_TYPE	An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- C32114A: An unterminated string literal occurs at line 62.
- B33203C: The reserved word "IS" is misspelled at line 45.
- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.
- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.
- B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.
- C41404A: The values of 'LAST and 'LENGTH are incorrect in the if statements from line 74 to the end of the test.
- B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type -- PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE -- at line 41.
- C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.
- B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and end case; is missing from line 42.
- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.
- B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.
- C87B50A: The call of "/=" at line 31 requires a use clause for package A.

- C92005A: The "/=" for type PACK.BIG_INT at line 40 is not visible without a use clause for the package PACK.

- C940ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- BC3204C: The body of BC3204C0 is missing.

END
DATE
FILMED
7-88
DTIC